IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re the Application of:	)	Group Art Unit: 2192
	)	
Bhavesh P. Davda	)	Examiner: Wei, Zheng
	)	
Serial No.: 10/650,257	)	Confirmation No.: 7452
	)	
Filed: August 27, 2003	)	
	)	
Atty. File No.: 4366-120	)	

For: "METHOD AND APPARATUS FOR HOT UPDATING OF RUNNING PROCESSES"

CERTIFICATE OF TRANSMISSION

I HEREBY CERTIFY THAT THIS CORRESPONDENCE IS BEING TRANSMITTED VIA THE OFFICE ELECTRONIC FILING SYSTEM IN ACCORDANCE WITH 37 CFR § 1.6(a)(4) ON

December 29, 2007

SHERIDAN ROSS P.C.

BY: _____

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P. O. Box 1450
Alexandria, VA 22313

APPELLANT'S BRIEF ON APPEAL (37 CFR § 41.31)

Dear Sir:

This is an appeal under 37 CFR § 41.31 to the Board of Patent Appeals and Interferences of the United States Patent and Trademark Office from the final rejection of Claims 1-21 of the above-identified patent application. These claims were indicated as finally rejected in an Office Action dated April, 9, 2007, and the Notice of Panel Decision from Pre-Appeal Brief Review dated November 30, 2007, indicated that these Claims remained rejected. Payment in the amount of $510.00 for the fee required under 37 CFR § 41.20(b)(2) is being submitted herewith via EFS-Web. Although Appellant believes this fee amount is correct and that no other fees are associated with this appeal, please charge any deficiency or credit any overpayment to Deposit Account No. 19-1970. The structure of the Brief is as follows in accordance with 37 CFR §41.37(c):

I. Real Party in Interest

II. Related Appeals and Interferences

III. Status of Claims

IV. Status of Amendments

V. Summary of Claimed Subject Matter

VI. Grounds of Rejection to be Reviewed on Appeal

VII. Argument

VIII. Claims Appendix

IX. Evidence Appendix- None

X. Related Proceedings Appendix- None

I. REAL PARTY IN INTEREST

Avaya Technology Corp. is the owner of the patent application and the real party in interest.

II. RELATED APPEALS AND INTERFERENCES

There are no other prior or pending appeals, interferences or judicial proceedings related to this patent application.

III. STATUS OF CLAIMS

The status of the claims is as follows:

1. Claims canceled: None.
2. Claims withdrawn from consideration but not cancelled: None.
3. Claims pending: 1-21.
4. Claims allowed: None.
5. Claims rejected: 1-21.
6. Claims objected to: None.

7. Claims appealed: 1-21.

IV. STATUS OF AMENDMENTS

An Amendment and Response that was filed on January 16, 2007 has been entered. An Amendment After Final was filed on June 8, 2007, within two months of the mailing date of the final rejection, however no Advisory Action was received by the Applicant. A Notice of Panel Decision from Pre-Appeal Brief Review was mailed on November 30, 2007, indicating that there is at least one actual issue for appeal, and requiring the filing of an Appeal Brief within one-month of the mailing date of the Notice. In accordance with 37 C.F.R. §41.37(c)(2), this brief does not include any new or non-admitted amendment.

V. SUMMARY OF CLAIMED SUBJECT MATTER (37 CFR §41.37(c)(1)(v))

The present invention is generally directed to updating running processes. In particular, embodiments of the invention set forth in independent Claims 1 and 10 generally provide for the replacement of a first instruction of a replaced function by a jump instruction and an address of an update table, in which the address of the update table contains an address of a first instruction of a first updated function in the program code. Claim 20 requires a debugging utility that stops execution of program code and determines the position of an instruction pointer, and a signal handler tool that replaces an address of a function to be replaced with an address of a replacement function in response to an instruction pointer being at least a predetermined distance from the address of the replacement function. The invention as claimed by independent Claim 18 requires stopping execution of program code and the replacement in memory of an address associated with a function to be replaced with the address of a replacement function. Accordingly, embodiments of the present invention allow for the execution of a replacement

function in place of a replaced function and can do so in connection with updating a running process.

More particularly, embodiments of the claimed invention relate to an executable program 204 that may be linked to one or more functions 208. (Specification p. 5, ll. 16-18; Fig. 2.) Each function 208 may be called during execution of the executable program 204, and the step of calling a function during execution of the executable program 204 may comprise directing a computer 100 to execute code contained at a particular memory address. (Specification p. 5, ll. 18-23.) By substituting an instruction directing the computer 100 to a memory address comprising an update table 212 in place of the first instruction of a function 208 that is being replaced, the executable program may be directed to an updated function 216. (Specification p. 6, ll. 1-4.) That is, the location within the update table 212 to which the computer 100 is directed in place of the first instruction of a replaced function 208 may contain an instruction to jump to a memory address containing the appropriate updated function 216. (Specification p. 6, ll. 4-7.) By replacing a first instruction associated with a function 208 with an instruction to execute a further instruction contained within an update table 212, a function 208 can be replaced by an updated function 216. (Specification p. 6, ll. 7-9.)

Operation of embodiments of the claimed invention in connection with installing a patched function in a running executable program include identifying the executable 204 to be patched. (Specification p. 7, ll. 18-19; Fig. 6.) Execution of the process is halted. According to at least some embodiments of the claimed invention the position of the instruction pointer is determined. (Specification p. 8, ll. 3-5.) Next, a determination is made as to whether the instruction pointer position is within X number of bytes from the address of the function or functions 208 to be patched. The number X is generally predetermined and is selected so that the running process can be updated glitchlessly. (Specification p. 8, ll. 5-8.)

If the instruction pointer is not within X bytes from the function 208 to be patched, the update table 212 is populated with the address of the new function 216. (Specification p. 8, ll. 9-11.) A jump instruction is then injected for the first instruction of the old function 204, followed by the address of the update table 212 entry corresponding to the new (or updated) function 216.

(Specification p. 8, ll. 13-15.) Accordingly, the address of the location in memory 112 corresponding to the entry within the update table 212 containing the address corresponding to the start of the updated function 216 replaces the first instruction of the original function 208. (Specification p. 8, ll. 15-18.) Execution of the process can then resume, and as a result of replacing the original function's 208 first instruction with a jump instruction that sends the running process to the update table 212, which in turn directs the running process to the updated function 216, the updated function 216 is executed in place of the original function 208. (Specification p. 8, l. 20 to p. 9, l. 2.)

Independent Claim 1 is generally directed to a method for updating a running process. The method includes:

allocating in executable program code 204 text first memory space 704 operable to receive new program instructions comprising at least a first updated function (Specification at p. 6, ll. 10-18; p. 9, l. 10 to p. 10, l. 9.);

allocating in executable program code 204 text second memory space 708 operable to receive address information related to said new program instructions 304 (Specification p. 6, l. 19 to p. 7, l. 9; p. 9, l. 10 to p. 10, l. 9.);

running said executable program code 204 (Specification p. 7, l. 18 to p. 8, l. 2.);

stopping execution of said executable program code 204, 620 (Specification p. 8, ll. 3-5.);

injecting a jump instruction and an address of an update table 212 at a location in a memory containing a first instruction 712 of a first replaced function 208,640, wherein said address of said update table 212 contains an address of a first instruction of said first updated function 216 (Specification p. 8, ll. 10-18.); and

resuming execution of said executable program code, wherein said first updated function 216 is called in place of said first function 208, 644, and wherein said executable code is updated in said memory (Specification p. 8, l. 19 to p. 9, l. 3.).

Independent Claim 10 is generally directed to a computer implemented method. The method includes:

receiving information identifying:

a running executable program 204 to be patched 404, 600, 604; and

a function to be replaced 208, 412 (Specification p. 3, ll. 1-5; p. 7, ll. 19-21.);

accessing a symbol table in a memory 112 for said executable program to be patched 416 (Specification p. 3, ll. 6-8; p. 7, ll. 3-6.);

obtaining from said symbol table an address of said function to be replaced 208, 416 (Specification p. 7, ll. 3-6.);

stopping execution by a processor 108 of said running executable program 204 to be patched 620 (Specification p. 8, ll. 3-5.);

injecting in said executable program 204 to be patched at a location in said memory 112 containing a first instruction of said function to be replaced 208 a jump instruction and an address of a new function 216, 640, wherein said new function 216 is executed by said processor 108 in place of said function to be replaced 208, and/or in a patched version of said executable program is created in said memory (Specification p. 2, ll. 17-21; p. 8, ll. 13-18; p. 10, lines 10-19); and

resuming execution of said executable program by said processor 108, wherein said patched version of said executable program is executed by said processor 644 (Specification p. 2, ll. 21-23; Specification p. 8, l. 19 to p. 9, l. 3.).

Independent Claim 18 is generally directed to a system for updating executable program code. The system includes:

means for receiving information identifying existing executable program code 204, 404. The means for receiving information identifying existing executable program code are provided by a create patch tool (Specification p. 3, ll. 3-5; p. 6, l. 19 to p. 7, l. 9; p. 7, ll. 18-22.).

means for receiving information identifying a function to be replaced 208, 412, 600, 604. These means are also provided by the create patch tool. (Specification p. 3, ll. 5-6; p. 7, ll. 3-9.).

means for stopping execution of said existing executable program code 204, 620. These means are provided by a debugger utility. (Specification p. 3, ll. 15-16; p. 8, ll. 3-5.).

means for inserting a jump code and an address associated with a replacement function 216 in place of an address of said function to be replaced 208 in said existing executable program code 204, wherein updated executable program code 204 is created. The means for inserting are provided by a signal handler tool (Specification p. 3, l. 9 to p. 4, l. 1; p. 7, ll. 10-17; p. 7, l. 22 to p. 8, l. 2.).

means for resuming execution of said executable program code 204, wherein said updated executable program code is executed. The means for resuming execution of the executable program code are provided by the debugger, which allows execution of the updated executable to resume after it is detached from the running process (Specification p. 8, ll. 19-20).

Claim 19 depends from Claim 20 and further recites jump table means, which are provided by the update or jump table (Specification p. 2, lines 18-21).

Independent Claim 20 is generally directed to a system for updating executing program code. The system includes:

a create patch tool operable to receive information identifying an executable program 204 to be updated and a function to be replaced 208 (Specification p. 3, ll. 3-8; p. 6, l. 20 to p. 7, l. 9; p. 7, ll. 19-21.);

a debugging utility operable to stop execution of said executable program 204 to be updated and to determine a position of an instruction pointer associated with said executable program 204 to be updated (Specification p. 3, ll. 15-16; p. 8, ll. 3-18.); and

a signal handler tool operable to replace in memory 112 an address of said function to be replaced 208 with an address of a replacement function 216 in response to said position of said instruction pointer being at least a predetermined distance from said address of said replacement function 216, wherein said replacement function 216 is executed instead of said function to be replaced 208 upon resuming execution of said executable program 204, wherein said executable program is updated (Specification p. 3, ll. 9-12; p. 7, ll. 10-17; p. 8, l. 10 to p. 9, l. 3; p. 19, ll. 1-14.).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL (37 CFR §41.37(c)(1)(vi))

A. The rejections under 35 U.S.C. §102(b), of Claims 1-5, 7, 10-13 and 16-19 as being anticipated by U.S. Patent No. 5,619,698 to Lillich et al.

B. The rejections under 35 U.S.C. §103 of Claims 6, 8, 14 and 15 as being unpatentable over Lillich in view of U.S. Patent App. Pub. No. 2004/0107416 to Buban et al.

C. The rejections under 35 U.S.C. §103 of Claim 9 as being unpatentable over Lillich in view of U.S. Patent App. Pub. No. 2003/0167463 to Munsil, et al.

D. The rejections under 35 U.S.C. §103 of Claims 20 and 21 as being unpatentable over U.S. Patent App. Pub. No. 2002/0152455 to Hundt, et al. in view of U.S. Patent App. Pub. No. 2004/0107416 to Buban et al.

VII. ARGUMENT

A. REJECTIONS UNDER 35 U.S.C. §102(b)

The Examiner has finally rejected Claims 1-5, 7, 10-13 and 16-19 as being anticipated by U.S. Patent No. 5,619,698 to Lillich et al. ("Lillich"). The relevant portions of 35 U.S.C. §102 provide that "a person shall be entitled to a patent unless - . . . (b) the invention was patented or described in a printed publication in this or a foreign country . . . , more than one year prior to the date of the application for patent in the United States." "A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described in a single prior art reference." (MPEP §2131 (quoting *Verdegaal Bros. v. Union Oil Co. of California*, 814 F 2d 628, 631, 2 USPQ 2d 1051, 1053 (Fed. Cir. 1987)). As explained below,

because each and every element of the claims appealed cannot be found in the Lillich reference, the Examiner's rejections under 35 U.S.C. §102 are improper and should be reversed.

1. The Lillich Reference

The Lillich reference is generally directed to a method and apparatus for patching operating systems. In the background discussion provided by Lillich, which has been cited by the Office Action in connection with the final rejections of the claims, a client call made to a replaced function is redirected to a patch comprising the computer code replacing the called function. In this conventional patch arrangement, an ATRAP instruction is included in application code. In response to reaching the ATRAP instruction, the microprocessor running the code will push the current state on to the computer's stack, and resume execution at the address indicated in a low memory location. (Lillich, col. 2, lines 48-54.) The indicated address is the address of the TRAP dispatcher. The TRAP dispatcher then operates to determine the operation indicated by the ATRAP instruction, looks up the address of the corresponding system routine in the TRAP table, and then jumps to the corresponding system routine. (Lillich, col. 3, lines 1-5.)

More particularly, the prior art discussed in Lillich and relied on by the Examiner with respect to the rejections makes us of a pre-existing ATRAP instruction 104 in the application code 102, which references a location in ATRAP table 110, via a low memory location 106, which is the address of the ATRAP dispatcher 108. (Lillich col. 2, l. 52 to col. 3, l. 1.) The ATRAP dispatcher 108 examines the bit pattern of the ATRAP instruction 104 to determine what operation it stands for, and looks up the address of the corresponding system routine in the TRAP table 110 (symbolized by line 144) and then jumps to the corresponding system routine (symbolized by line 146 in Fig. 2 of Lillich). (Lillich col. 3, ll. 1-6 and ll. 15-25.) Accordingly, the system code 122 represents an original or unpatched system routine. (Lillich col. 3, ll. 6-14.)

According to Lillich, in order to redirect to a patch system routine, the TRAP dispatcher 108 will perform the same lookup as before (*i.e.*, the same address in the TRAP table 110 that

was used to direct the process to the system routine prior to implementing the patch). "However, in the patched case, address _ 1 will point to patch code 132 located in RAM 130 rather than the original system routine." (Lillich col. 3, ll. 44-46.) Therefore, the address accessed by the TRAP dispatcher 108 is changed from the address of the original system routine 122 to the patch code 132, and the process jumps to the patch code 132 (symbolized by line 150 in Fig. 2 of Lillich). Accordingly, it is clear that Lillich does not inject an instruction at a location in memory containing a first instruction of a function to be replaced, for example as required by Independent Claims 1 and 10. In addition, Lillich does not provide means for stopping execution of code, means for inserting a jump code and an address associated with a replacement function, means for resuming execution of code as recited by independent Claim 18. Lillich also does not provide a debugging utility that is operable to determine a position of a instruction pointer associated with the executable program or a signal handler tool as recited by Claim 20.

2. Independent Claims 1 and 10 and Dependent Claims 2-9 and 11-17 are not anticipated by Lillich

The invention set forth in Claims 1-17 is generally directed to methods for updating a running process. More particularly, a running process is updated by injecting a jump instruction and an address of an updated or new function at a location in memory containing a first instruction of a first replaced function or function to be replaced. At least this aspect of the claims is not present in the Lillich reference.

Instead, Lillich changes an address in a TRAP table in order to change the code that is executed next. In particular, when the application code reaches a trap instruction, Lillich accesses the TRAP table to look up the address of the corresponding system routing. In order to update or replace an existing system routing, Lillich specifies that the address in the TRAP table 110 for the TRAP instruction be changed from the address of the replaced system routine to the address of the new system routine. Of course, this is quite different from injecting a jump instruction and an address at a location in memory containing a first instruction of a first replaced

function, as required by Claim 1, or injecting at a location in memory containing a first instruction of the function to be replaced a jump instruction and an address of a new function, as recited by Claim 10.

Accordingly, it can be appreciated that the prior art patching paradigm discussed by Lillich does not describe each and every element and rejection of these claims as anticipated by Lillich should be reconsidered and withdrawn.

3. Independent Claim 18 and Dependent Claim 19 are not Anticipated by Lillich

The invention set forth in Claims 18-19 is generally directed to a system for updating executable program code. More particularly, the system includes means for stopping execution of the existing executable program code, means for inserting a jump code and an address associated with a replacement function in place of an address of the function to be replaced in the existing executable program code, and a means for resuming execution of the executable program code, wherein the updated executable program code is executed. These aspects of Claims 18 and 19 are not present in the Lillich reference.

The final action, with respect to Claims 18 and 19, contains the cursory statement that these claims would also be anticipated by Lillich, because they are system claims for updating executable program code using the methods discussed in Claims 10-11. Initially, Applicants note that the elements of Claim 18 do not necessarily correspond to those of Claim 10. In addition, it is not apparent that Lillich is directed to patching program code while that code is executing. This is highlighted by the omission by the applicable portions of Lillich of any mention of stopping execution of code to perform at patch, performing the patch operation, and resuming execution of program code as claimed. Accordingly, the rejections of Claims 18 and 19 are unsupported, and the elements of these claims are not disclosed by the Lillich reference. Accordingly, the rejections of Claims 18 and 19 as anticipated should be reconsidered and withdrawn.

B. REJECTIONS UNDER 35 U.S.C. § 103

Pending Claims 6, 8, 14 and 15 have been finally rejected as being unpatentable over Lillich in view of Patent Application Publication No. 2004/0107416 to Buban et al ("Buban"), Claim 9 has been finally rejected as being unpatentable over Lillich in view of U.S. Patent Application Publication No. 2003/0167463 to Munsil et al ("Munsil"), and Claims 20 and 21 have been finally rejected as being unpatentable over U.S. Patent Application Publication No. 2002/0152455 to Hundt et al ("Hundt") in view of Buban. 35 U.S.C. § 103 provides in relevant part:

> A patent may not be obtained though the invention is not identically disclosed or described as set forth in § 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains.

35 U.S.C. § 103

> A *prima facie* case of obviousness is established [by an Examiner] when the teachings from the prior art itself would appear to have suggested the claimed subject matter to a person of ordinary skill in the art.

In Re *Rijckaert,* 28 USPQ 2d (BNA) 1955, 1956 (quoting In Re *Bell*, 26 USPQ 2d (BNA) 1529, 1531 (Fed. Cir. 1993)). "[R]ejections on obviousness cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness." *KSR International Co. v. Teleflex Inc.*, 550 U.S. ____, 82 USPQ2d 1385, 1396 (2007).

In determining the propriety of the Patent Office case for obviousness in the first instance, it is necessary to ascertain whether or not the reference teachings would appear to be sufficient for one of ordinary skill in the art having the reference before him to make the

proposed substitution, combination, or other modification. (MEPE, § 2143.01, quoting In Re *Linter*, 458 F.2d 1013, 1016 (CCPA 1972).

1. Dependent Claims 6, 8, 14 and 15 are not Obvious Over Lillich In View of Buban

Claims 6, 8, 14 and 15 are dependent claims that generally specify determining a distance or number of bytes between a position within program code at which execution is stopped and an address of a function to be updated or replaced. For disclosure of determining a distance between memory addresses, the final Office Action cites to the Buban reference. However, such disclosure is, in fact, completely absent from Buban. In particular, paragraphs 46 and 47 of Buban, which have been cited by the final Office Action, make reference to patching running code and specifies that the replaced instruction may need to be completely contained within a processor's smallest unit of atomically replaceable memory. (Buban, paragraph 46). However, there is no mention in the cited portions of Buban of determining a distance between a position within code where execution is stopped and an address of a function to be updated or replaced. Accordingly, the rejections of Claims 6, 8, 14 and 15 should be reconsidered and withdrawn for at least this reason. Moreover, Buban does not make up for the deficiencies of the Lillich reference with respect to the independent claims from which Claims 6, 8, 14 and 15 depend. Therefore, Claims 6, 8, 14 and 15 should be allowed for at least this additional reason.

2. Dependent Claim 9 is not Obvious Over Lillich In View of Munsil

Claim 9 depends from Claim 1 and specifies that the second memory space is read-only memory space. As recited by Claim 1, the second memory space is operable to receive address information related to the new program instructions. For a disclosure of the use of read-only memory, the Office Action cites to Munsil. However, Munsil does not make up for the deficiencies of the Lillich reference with respect to independent Claim 1. Therefore, for at least

the reason that each and every element of Claim 1 is not taught, suggested or described by the cited references, the rejection of Claim 9 as obvious should be reconsidered and withdrawn.

3. Independent Claim 20 and Dependent Claim 21 are not Obvious Over Hundt In View of Buban

Independent Claim 20 is generally directed to a system for updating executing program code. The system includes a signal handler tool operable to replace in memory an address of the function to be replaced with an address of a replacement function. The replacement is performed in response to the instruction pointer being at least a predetermined distance from the address for the replacement function.

The Hundt reference discusses using original functions to generate instrumented functions in executing the instrumented functions in place of the original functions. (See Hundt, paragraph 18). However, there is no disclosure in Hundt of replacing in memory an address of a function to be replaced with an address of a replacement function in response to the instruction pointer being at least a predetermined distance from the address of the replacement function. The Office Action acknowledges that there is no disclosure in Hundt related to making such a replacement in response to the position of the instruction pointer being at least a predetermined distance from the address of the replacement function. For such a disclosure, the Office Action cites to Buban. However, Buban, in fact, contains no teach, suggestion or disclosure of such a predetermined distance. Therefore, the rejections of Claims 20 and 21 should be reconsidered and withdrawn.

Based upon the foregoing, Appellant respectfully requests that the Board reverse the Examiner's rejections of pending Claims 1-21and requests that the board pass the above-identified patent application to issuance.

Respectfully submitted,

SHERIDAN ROSS P.C.

By: _____

Bradley M. Knepper
Registration No. 44,189
1560 Broadway, Suite 1200
Denver, Colorado 80202-5141
(303) 863-9700

Date: December 29, 2007

VIII. CLAIMS APPENDIX

1. A method for updating a running process, comprising:

allocating in executable program code text first memory space operable to receive new program instructions comprising at least a first updated function;

allocating in executable program code text second memory space operable to receive address information related to said new program instructions;

running said executable program code;

stopping execution of said executable program code;

injecting a jump instruction and an address of an update table at a location in a memory containing a first instruction of a first replaced function, wherein said address of said update table contains an address of a first instruction of said first updated function; and

resuming execution of said executable program code, wherein said first updated function is called in place of said first replaced function, and wherein said executable code is updated in said memory.

2. The method of Claim 1, wherein said step of resuming execution of said executable program code comprises running an intermediate executable, wherein said intermediate executable comprises said updated copy of said executable program code in said memory.

3. The method of Claim 1, further comprising:

before said step of running said executable program code, copying said executable program code to said memory.

4. The method of Claim 1, further comprising:

injecting a jump instruction and an address of said update table at a location in a stored copy of said executable program code containing an address of said first replaced function; and

populating said update table with an address of said first updated function, wherein a stored copy of said executable program code is updated.

5. The method of Claim 4, wherein said updated stored copy of said executable program code comprises final updated executable program code.

6. The method of Claim 1, further comprising:

determining a first distance between a position within said code text at which execution of said executable program code is stopped and an address of a first function, wherein said first function is a function to be updated; and

in response to said first distance exceeding a predetermined amount, populating an update table stored in memory with an address of a first updated function.

7. The method of Claim 1, wherein said injected jump instruction replaces a first instruction of said first replaced function.

8. The method of Claim 6, wherein said predetermined amount is 8 bytes.

9. The method of Claim 1, wherein said second memory space is read only memory space.

10. A computer implemented method, the method comprising:

receiving information identifying:

a running executable program to be patched; and

a function to be replaced;

accessing a symbol table in a memory for said executable program to be patched;

obtaining from said symbol table an address of said function to be replaced;

stopping execution by a processor of said running executable program to be patched;

injecting in said running executable program to be patched at a location in said memory containing a first instruction of said function to be replaced a jump instruction and an address of a new function, wherein said new function is executed by said processor in place of said function to be replaced, and wherein a patched version of said executable program is created in said memory; and

resuming execution of said executable program by said processor, wherein said patched version of said executable program is executed by said processor.

11. The method of Claim 10, further comprising providing a jump table in said memory, wherein said injecting in said running executable program to be patched a jump instruction and an address of a new function comprises replacing a first instruction line associated with said function to be replaced with an instruction to jump to a first address contained within said jump table, and wherein said first address contained within said jump table comprises an address of a first instruction line of said new function.

12. The method of Claim 11, wherein said first instruction line associated with said function to be replaced comprises a first instruction of said function to be replaced, wherein said first instruction is not an instruction to jump to another address, and wherein said first instruction of said function to be replaced is replaced by said instruction to jump to a first address.

13. The method of Claim 11, wherein said first instruction line associated with said function to be replaced comprises an instruction to jump to a second address contained within said jump table, and wherein said instruction to jump to a second address is replaced by said instruction to jump to a first address.

14. The method of Claim 10, further comprising:

determining a number of bytes between a location within said executable program at which said executable program is stopped and an address of said function to be replaced.

15. The method of Claim 14, further comprising:

in response to said determined number of bytes being at least as great as a first selected number, injecting in a stored copy of said running executable program to be patched said jump instruction and said address of said new function in place of said address of said function to be replaced, wherein a patched version of said executable program is created.

16. The method of Claim 10, wherein said computational component comprises a computer readable storage medium containing instructions for performing the method.

17. The method of Claim 10, wherein said computational component comprises a logic circuit.

18. A system for updating executable program code, comprising:

means for receiving information identifying existing executable program code;

means for receiving information identifying a function to be replaced;

means for stopping execution of said existing executable program code;

means for inserting a jump code and an address associated with a replacement function in place of an address of said function to be replaced in said existing executable program code, wherein updated executable program code is created; and

means for resuming execution of said executable program code, wherein said updated executable program code is executed.

19. The system of Claim 18, further comprising jump table means, wherein said address associated with a replacement function comprises an address within said jump table containing an address of a first instruction of said replacement function.

20. A system for updating executing program code, comprising:

a create patch tool operable to receive information identifying an executable program to be updated and a function to be replaced;

a patch tool operable to query an operating system for a process identifier associated with said identified executable program;

a debugging utility operable to stop execution of said executable program to be updated and to determine a position of an instruction pointer associated with said executable program to be updated; and

a signal handler tool operable to replace in memory an address of said function to be replaced with an address of a replacement function in response to said position of said instruction pointer being at least a predetermined distance from said address of said

replacement function, wherein said replacement function is executed instead of said function to be replaced upon resuming execution of said executable program, wherein said executable program is updated.

21. The system of Claim 20, wherein said signal handler is additionally operable to replace in a stored copy of said executable program an address of said function to be replaced with an address of a replacement function, wherein said stored copy of said executable program is updated.

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.